VLSI Array Processors for Linear-Phase FIR Filters

Esam Abdel-Raheem, Fayez El-Guibaly, and Andreas Antoniou
Department of Electrical and Computer Engineering
University of Victoria, P.O. Box 3055,
Victoria, B.C., Canada, V8W 3P6
email: fayez@sirius.uvic.ca

Abstract

Array processor implementations are obtained for linear-phase FIR filters. Three structures are reported in which the inputs are pipelined and/or broadcast and the outputs are pipelined. A novel structure is obtained in which the outputs are localized in separate processing elements. A comparison among the resulting structures is performed based on the sampling rate, the latency, and the communication overhead perspectives. A new fixed-point array-multiplier design is then presented. The new processor can perform an add-multiply-accumulate operation in the same time as a simple multiplier. It increases the speed of operation without incurring extra silicon area or introducing extra latency to the system.

1 Introduction

Real-time signal processing can be achieved using special-purpose array processors, and maximizing processing concurrency though pipeline processing [1]. Array processors are especially suitable to a special class of compute-bound iterative algorithms, taking advantage of their regular, and sometimes localized data flow. Consequently, they have been used effectively in the area of digital filtering, digital signal processing, and matrix algebra [1]-[3]. Multiplier and adder delays are the dominant factors which determine the speed of the array structure. Minimizing these delays results in a high-speed array processor suited for high-speed digital signal processing applications.

Linear phase is obtained in FIR filters if the finite-duration impulse response has even or odd symmetry about its midpoint [4], i.e.,

$$h(nT) = \pm h[(N-1-n)T], \quad 0 \le n < N$$

where N is the filter length, and T is the sampling period. The positive and negative signs represent the even and odd symmetries, respectively. Using this condition, we can write

$$y(nT) = \sum_{k=0}^{N-1} h(kT)x(nT-kT) \quad n \ge 0$$
$$= \sum_{k=0}^{N/2-1} h(kT)[\mathcal{E}^{-k}x(nT) \pm \mathcal{E}^{-(N-1-k)}x(nT)]$$

for N even, and

$$y(nT) = \sum_{k=0}^{(N-1)/2} a(kT)[\mathcal{E}^{-k}x(nT) \pm \mathcal{E}^{-(N-1-k)}x(nT)]$$

for N odd, where $a(nT) = h(nT)$, for $n = 0, 1, \ldots, (N-3)/2$, and $a[(N-1)T/2] = 0.5h[(N-1)T/2]$. $\mathcal{E}$ is the shift operator defined by [4]

$$\mathcal{E}^{-k}x(nT) = x(nT-kT)$$

Note that for N odd, $h[(N-1)T/2] = 0$ for the anti-symmetrical case.

In this paper, array processor implementations for linear-phase FIR filter algorithms are obtained using the approach in [1]. The resulting filter structures are modular and pipelined. Three different schemes are reported. In the first scheme, the input and output are pipelined. In the second scheme, the inputs are pipelined and broadcast and the partial results are pipelined. In the third scheme, each partial output stays in a separate processing element (PE) to accumulate its terms thereby allowing efficient use of a new multiplier-accumulator hardware. A special fixed-point processor module based on the parallel multiplier design is presented. The processor performs an add-multiply-accumulate operation in the same time as a simple multiplier. The module enhances the speed of operation without incurring extra silicon area or introducing extra latency to the system.

1058-6393/95 $4.00 © 1995 IEEE

2 Array Processor Implementation of Linear-Phase FIR Filters

The array processors presented will be for the symmetrical, odd-length filters. Structures for antisymmetrical and even-length filters can be deduced in a similar way.

The design of an array structure for a given algorithm can be decomposed into three stages [1]: (a) mapping the algorithm to the dependence graph (DG) describing the dependencies of the data, (b) converting the DG to the signal flow graph (SFG). This is done by choosing a projection vector d and a schedule vector s, and (c) mapping the SFG onto an array processor.

The DG of linear-phase FIR-filter algorithms can be derived by using a DG for a general FIR filter algorithm and folding it about its midline related to the midpoint in the coefficient space [1]. Applying schedule vector s gives rise to equitemporal hyperplanes along which all computations are performed at the same clock cycle. The next step is to apply a projection vector d which determines the array configuration. All nodes lying along a straight line parallel to d are assigned to the same PE [1].

2.1 Scheme I

An array processor in which the inputs and the outputs are pipelined can be obtained if a schedule vector $s = [1 \quad -2]^T$ is to be employed. The DG for $N = 5$ is shown in Fig. 1(a) where the inputs $x(nT)$ and $w(nT) = x(nT)$ and the output $\hat{y}(nT) = \mathcal{E}^{-(N-1)/2}y(nT)$ are pipelined through the index space (n, k). Projecting the DG along the projection vector $d = [1 \quad 0]^T$, results in the array processor of Fig. 1(b). The details of the PE involved are shown in Fig. 1(c). A similar scheme has been reported in [5].

2.2 Scheme II

It is possible to obtain an array structure in which the inputs are broadcast/pipelined and the outputs are pipelined. Choosing a schedule vector $s = [1 \quad -1]^T$, the modified DG in Fig. 2(a) is produced where the output $y(nT)$ and input $w(nT)$ are pipelined and the input $x(nT) = w(nT)$, $n = 0, 1, \ldots$, is broadcast. Projecting the DG along the projection vector $d = [1 \quad 0]^T$, the linear array processor in Fig. 2(b) is produced. Figure 2(c) shows the details of the PE involved.

2.3 Scheme III

Figure 3(a) repeats the DG of Fig. 2(a). Projecting the DG along the projection vector $d = [0 \quad -1]^T$ results in an array processor that has an infinite number of PEs, which is not practical. However, this impracticality can be eliminated by recognizing the fact that the first and second PEs are idle after one and two sampling periods, respectively, and every other PE is idle after 3 sampling periods ($(N + 1)/2$ sampling periods in general). Thus, processing element PE(i) that computes $y(iT)$ can be used again to compute $y[(i + nl)T]$ where $l = (N + 1)/2$; hence the number of PEs in the array, N_p, is equal to $(N + 1)/2$.

To obtain the array structure, a data flow/timing table is constructed in Table 1 based on the data flow and timing in the DG of Fig. 3(a). Mapping the data flow/timing information available in Table 1 onto an array structure results in the array processor scheme in Fig. 3(b) where all the data shown correspond to the sampling instant T_0 in Table 1. The control select signal for each multiplexer block is implemented by means of a ring memory. The multiplexers are used to update the signal $w(nT)$, $n = 0, 1, \ldots$, according to Table 1. Also, another ring memory is implemented in each PE for updating the filter coefficient for each sampling period according to Table 1. Using ring memories is preferred over transmitting the control select signal and the filter coefficients across different PEs to reduce the communication overhead. The ouput is provided through an output bus which is driven by a set of three-state drivers. One driver is enabled at a time in a round robin fashion so each PE provides an output once every $(N + 1)/2$ sampling periods.

3 Performance Analysis

The filter maximum sampling rate depends on the hardware details. For the basic direct-form linear-phase FIR scheme [6] in which the inputs are pipelined and the outputs are added simultaneously, the minimum sampling period is determined by the time required to produce the filter output. Clearly, such an implementation is practical only for low-order filters where the asynchronous adder does not dominate the response time of the system.

For scheme I, the minimum sampling period is determined by the delay in one pipeline stage. In this case, the above sampling time does not depend on the number of PEs involved which makes this scheme practical for high-order filters. Moreover, the full local communication feature makes the scheme attractive for high-speed signal processing applications.

For schemes II and III, the minimum sampling period is determined by the delay in one pipeline stage and the data bus delay. The bus delay is proportional to the square of the number of PEs (i.e., N_p^2) [7] and may impose an upper bound on the sampling rate for high-order filters.

System latency is defined as the time elapsed between the application of a sample to the input and the appearance of the corresponding sample at the output. The latency for the basic direct-form scheme [6] is one sampling period T. The latency for scheme I is N_pT while that for schemes II and III is T.

4 System Speedup through Hardware Improvement

In this section, a hardware improvement to increase the sampling rate is presented. Efficient designs of parallel multipliers for 2's-complement arithmetic are reported in [8]-[10]. In [10], an accumulator-multiplier (AM) processor is reported which is based on the regular design of a parallel multiplier discussed in [8]. The improved processor that will be discussed is based on the accumulator-multiplier processor reported in [10].

Figure 4(a) shows the hardware details of each PE for the array implementation scheme discussed previously. We refer to this conventional design as an adder-multiplier-accumulator (AMA) module. The delay in each PE is the sum of two adder delays and one multiplier delay. One attempt to reduce the PE delay is to use an add-accumulate multiply (AAM) module as in Fig. 4(b) in which an adder is followed by the AM processor reported in [10]. It is possible, however, to reduce the PE delay to just the multiplier delay without incurring extra hardware penalties by performing the addition and multiplication operations in the same time. Figure 4(c) shows the new merged-operand (MO) module that achieves this performance improvement. The details of a 4×4 MO module are illustrated in Fig. 5. It is noted that the addition of the two operands x and w is performed simultaneously with the multiplication operation.

In the following, the propagation delay and the area of the MO module will be compared to those of the AMA and AAM modules. The propagation delay of the standard design of Fig. 4(a) is given by

$$T_{AMA} \approx T_{add-sp} + T_{mult} + T_{add-dp} + T_l \qquad (1)$$

where T_{add-sp} is the delay of an n-bit adder, T_{mult} is the delay of an $n \times n$ 2's-complement multiplier, T_{add-dp} is the delay of a $2n$-bit adder, and T_l is the delay of a 1-bit latch. Alternatively, the above formula can be expressed as

$$T_{AMA} \approx 5(n-1)T_{FA} + 3T_{HA} + T_A + T_{inv} + T_l \qquad (2)$$

where T_{FA}, T_{HA}, T_A, T_{inv} are the propagation delays for a 1-bit full-adder, a 1-bit half-adder, a 2-input AND gate, and a 1-bit inverter, respectively. Similarly, the propagation delay of the design of Fig. 4(b) is given by

$$T_{AAM} \approx (3n-2)T_{FA} + 2T_{HA} + T_A + T_l \qquad (3)$$

The propagation delay of the MO module is determined by the time required for the longest path indicated by the dotted line in Fig. 5. It is given by

$$T_{MO} \approx (2n-1)T_{FA} + 2T_{HA} + T_A + T_l \qquad (4)$$

The area required by the design of Fig. 4(a) can be expressed as

$$\begin{aligned} A_{AMA} \approx\ & [n^2+n-1]A_{FA} + (n+1)A_{HA} \\ & + [(n-1)^2+1]A_A + 2(n-1)A_N \\ & + A_{inv} + 2nA_l \end{aligned} \qquad (5)$$

where A_{FA}, A_{HA}, A_A, A_N, A_{inv}, and A_l are the layout areas for a 1-bit full-adder, a 1-bit half-adder, a 2-input AND gate, a 2-input NAND gate, a 1-bit inverter, and a 1-bit latch, respectively. It is easy to verify that the areas of the AAM and MO modules are approximately equal to that of the AMA module; hence they can be represented by the above formula. It is noted that in deriving the above relations, we assumed the routing areas to be the same for the three modules.

It can be seen that the above computations depend on several factors such as the type of technology implemented and the layout style. In order to compare the delay performance of the above two designs, some normalization is carried out. We can normalize all delays relative to the inverter delay, T_{inv}, based on the following assumptions [7],[11]: $T_A \approx 2T_{inv}$, $T_{HA} \approx 5T_{inv}$, $T_{FA} \approx 6T_{inv}$, and $T_l \approx 4T_{inv}$. Referring to (2)-(4), the increase in speed of the AAM and the MO modules over the AMA module can be expressed in terms of speedup gains S_{g1} and S_{g2} defined as

$$S_{g1} = \frac{T_{AMA}}{T_{AAM}} = \frac{30n-8}{18n+4}, \quad S_{g2} = \frac{T_{AMA}}{T_{MO}} = \frac{30n-8}{12n+10}$$

Figure 6 shows the two speedup gains versus the data wordlength. It is evident that the proposed design offers the most improvement.

Generally, the MO module can be incorporated in schemes I and II to perform an add-multiply-accumulate process as well as in scheme III to perform an inner-product processing.

5 Comparisons and Discussion

In the previous sections, array processor implementations with hardware improvements have been obtained. From the perspective of speed, higher data rates can be achieved with scheme I than with schemes II and III especially for high-order filters. In scheme III, the output word can be truncated (or rounded) locally in each PE. This would decrease the communication overhead compared to that in schemes I and II in which the output word is to be truncated in the final PE if noise performance is to be improved. From the perspective of latency, schemes II and III have the minimum latency of one sampling period.

It is pointed out in [6] that the cascade form is generally less sensitive to coefficient quantization than the equivalent direct form realization. If $H(z)$ is the

z transform of $h(nT)$, then the symmetry condition causes the zeros of the linear-phase transfer function $H(z)$ to occur in mirror-image pairs [4]. Zeros at $z = \pm 1$ are their own reciprocal. Complex zeros on the unit circle are conveniently grouped in pairs. Complex zeros not on the unit circle occur in groups of four. Therefore, the system can be implemented as a cascade of first-, second-, and fourth-order systems. Each of these systems is a polynomial whose coefficients have the same symmetry as the coefficients of $H(z)$. Hence, the array processor schemes obtained in section 2 can be used to implement these systems.

6 Conclusions

Array processors for linear-phase FIR filters have been obtained using the signal flow graph approach. The method was employed to obtain array structures for odd-length filters. Even-length filter structures can be obtained in a similar way using the same PEs as for the odd-length filter arrays. All the implementations obtained are modular and regular. A comparison among various schemes was provided. A new processor module has been presented which enhances the speed of operation without increasing the silicon area or introducing extra latency in the system.

Acknowledgment

The authors wish to thank Micronet, Canadian Networks of Centres of Excellence Program, and the Natural Sciences and Engineering Research Council of Canada for supporting this work.

References

[1] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, 1988.

[2] F. El-Guibaly, S. Sunder, and A. Antoniou, "Systolic implementation of FIR filters", *Signal Processing V: Theories and Applications*, Elsevier Science Publishers, pp. 1423-1426, Sept. 1990.

[3] S. Sunder, F. El-Guibaly, and A. Antoniou, "Systolic implementation of digital filters", *Multidimensional Systems and Signal Processing*, vol. 3, pp. 63-78, Mar. 1992.

[4] A. Antoniou, *Digital Filters: Analysis, Design, and Applications*, 2nd ed., McGraw-Hill, 1993.

[5] H. K. Kwan and T. S. Okullo-Oballa, "Systolic realization of linear-phase FIR digital filters", *IEEE Trans. Circuits, Sys.*, vol. CAS-34, pp. 1604-1605, Dec. 1987.

[6] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, 1989.

[7] N. E. Weste and K, Eshraghian, *Principles of CMOS VLSI Design*, 2nd Ed., Addison-Wesley, 1993.

[8] M. Hatamian and G. L. Cash, "A 70-MHz 8×8 parallel pipelined multiplier in 2.5 μm CMOS", *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 505-513, Aug. 1986.

[9] C. P. Lerounge, P. Girard, and J. S. Colardelle, "A fast 16 bit NMOS parallel multiplier", *IEEE J. Solid-State Circuits*, vol. SC-19, pp. 338-342, June 1984.

[10] S. Sunder, F. El-Guibaly and A. Antoniou, "VLSI implementation of a second-order digital filter", *Can. J. Elec. & Comp. Eng.*, vol. 19, No. 3, 1994.

[11] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley, 1979.

Table 1: Data flow/timing for scheme III (assuming $T = 1$ sec.).

T	PE(0)			PE(1)			PE(2)		
	$x(n)$	$a(k)$	$w(n)$	$x(n)$	$a(k)$	$w(n)$	$x(n)$	$a(k)$	$w(n)$
T_0	$x(0)$	$a(0)$	0	$x(0)$	$a(1)$	0	$x(0)$	$a(2)$	$w(0)$
T_1	$x(1)$	$a(2)$	$w(1)$	$x(1)$	$a(0)$	0	$x(1)$	$a(1)$	0
T_2	$x(2)$	$a(1)$	$w(0)$	$x(2)$	$a(2)$	$w(2)$	$x(2)$	$a(0)$	0
T_3	$x(3)$	$a(0)$	0	$x(3)$	$a(1)$	$w(1)$	$x(3)$	$a(2)$	$w(3)$
T_4	$x(4)$	$a(2)$	$w(4)$	$x(4)$	$a(0)$	$w(0)$	$x(4)$	$a(1)$	$w(2)$
T_5	$x(5)$	$a(1)$	$w(3)$	$x(5)$	$a(2)$	$w(5)$	$x(5)$	$a(0)$	$w(1)$
T_6	$x(6)$	$a(0)$	$w(2)$	$x(6)$	$a(1)$	$w(4)$	$x(6)$	$a(2)$	$w(6)$
T_7	$x(7)$	$a(2)$	$w(7)$	$x(7)$	$a(0)$	$w(3)$	$x(7)$	$a(1)$	$w(5)$
T_8	$x(8)$	$a(1)$	$w(6)$	$x(8)$	$a(2)$	$w(8)$	$x(8)$	$a(0)$	$w(4)$
T_9	$x(9)$	$a(0)$	$w(5)$	$x(9)$	$a(1)$	$w(7)$	$x(9)$	$a(2)$	$w(9)$

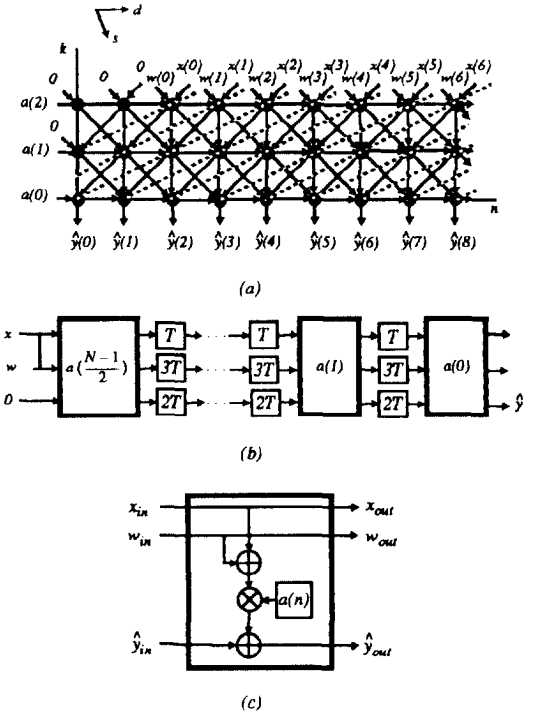

Figure 1: (a) DG for scheme I (assuming $T = 1$ sec.). (b) Scheme I array processor. (c) Details of the PE involved.

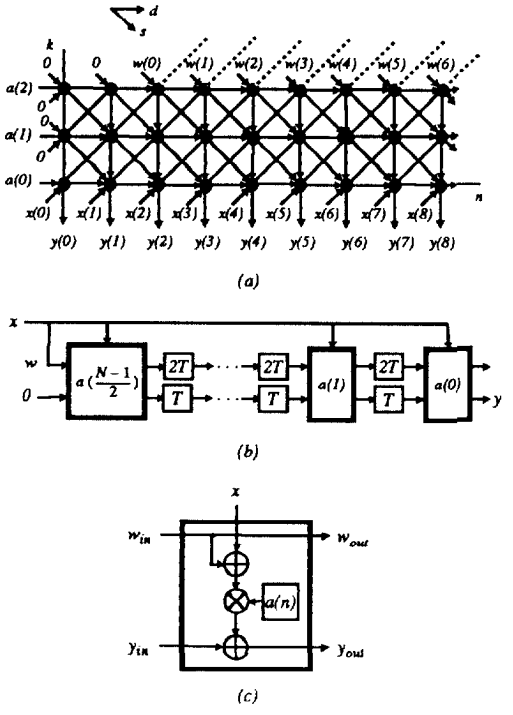

Figure 2: (a) DG for scheme II (assuming $T = 1$ sec.). (b) Scheme II array processor. (c) Details of the PE involved.

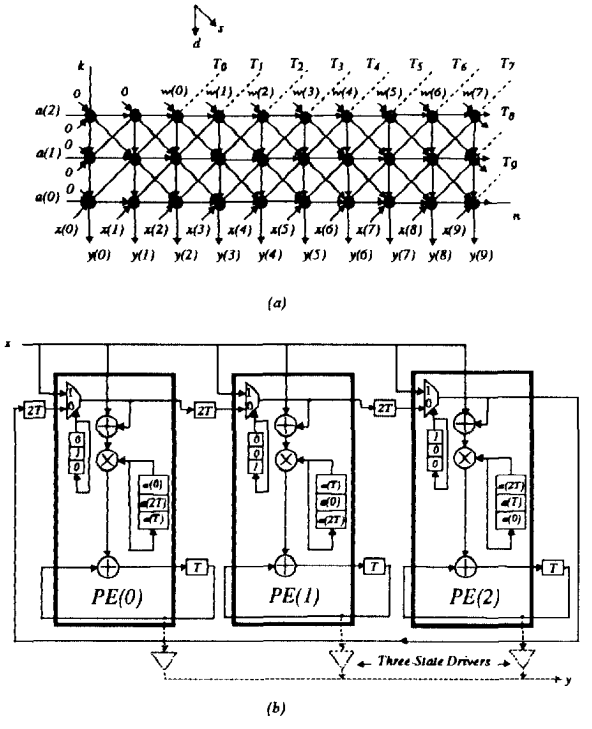

Figure 3: (a) DG for scheme III (assuming $T = 1$ sec.). (b) The array processor.

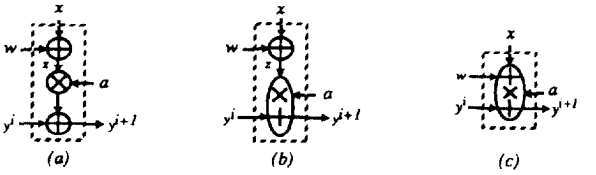

Figure 4: (a) Adder-multiplier-accumulator (AMA) module. (b) Adder-accumulator-multiplier (AAM) module. (c) Merged-operand (MO) module.

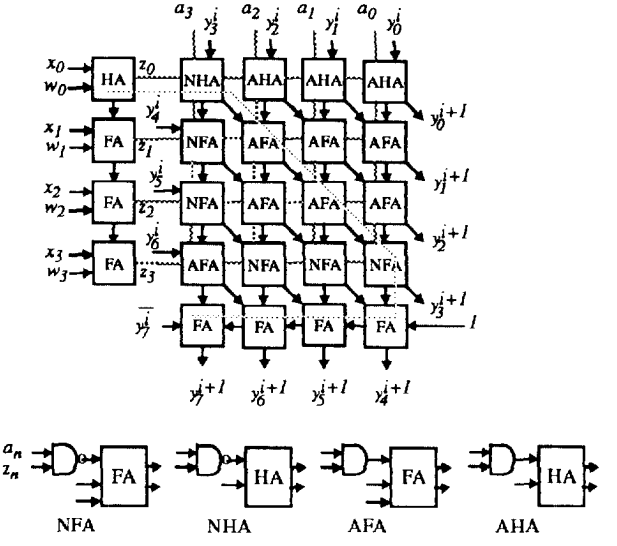

Figure 5: Details of a 4×4 2's-complement merged-operand (MO) module.

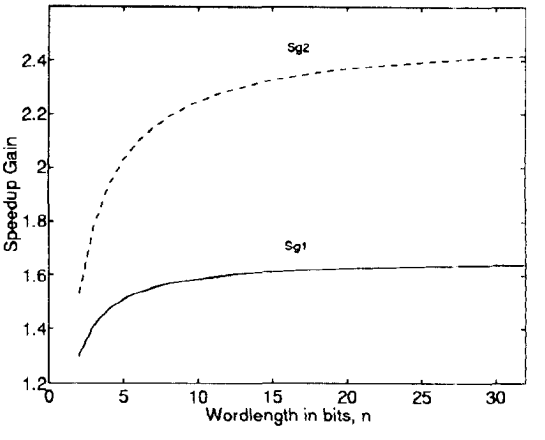

Figure 6: Speedup gain versus wordlength.